Introduction to Word Embeddings

307307 BI Methods

Converting Words to Numbers

Old Method:- One-Hot Encoding

Each word is represented by setting one dimension to 1 and all the other dimensions to ZEROS.

id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding

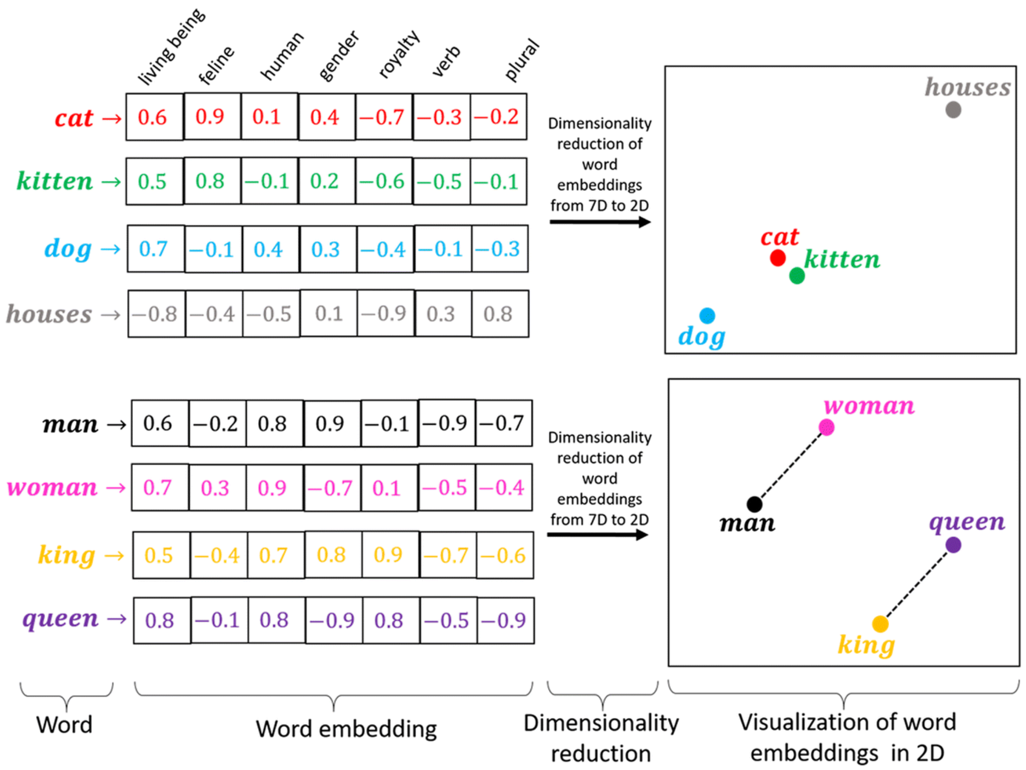
id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Limitations:

- Vocabulary of 100,000 words → 100,000 dimensions [0010000000000000000000000000000000000000…etc.]
- All words equally distant from each other
- No semantic meaning captured

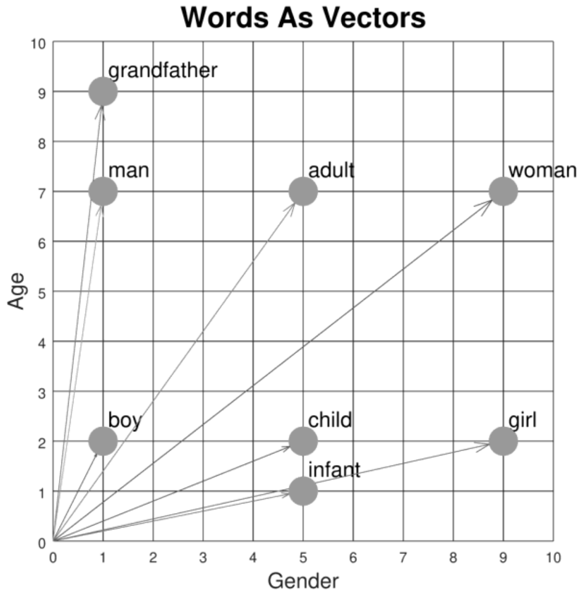
Word Embeddings and Vector Spaces

- Words are encoded as dense numerical vectors instead of one-hot or sparse representations.
- Similar words → similar vectors
- Typically, 50-300 dimensions (vs. vocabulary size in 1-hot-encoding)
- Word Embeddings captures semantic and syntactic relationships about/between words.
- Each dimension potentially captures some syntactic or semantic meaning.
- These vectors are learned from text by models like Word2Vec.



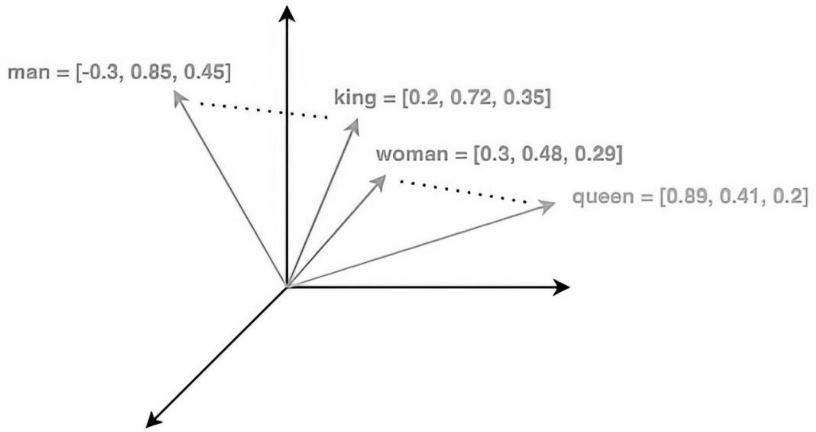
Vector Spaces and Word Embeddings

- A vector space is a mathematical space where each word is represented as a point (or vector) in multi-dimensional space.
- Makes it possible to compare, visualize, and manipulate meanings of words using math.
- Enables operations like:
  - Similarity: "king" is close to "queen"
  - Analogy: "king" - "man" + "woman" ≈ "queen"



Properties of Vector Space

- Semantic relationships are preserved (e.g., “man" is closer to “grandfather" than “girl").
- Closer Vectors → Similar Meanings
- Vectors Farther Apart → Dissimilar Meanings



Key Intuition

- Distributional hypothesis - "You shall know a word by the company it keeps" (J.R. Firth, 1957)
- Words in similar contexts have similar meanings:
- "The **cat** sat on the mat"
- "The **dog** sat on the mat"
- "The **rabbit** sat on the mat"
- → cat, dog, rabbit learn similar representations

Word2Vec (Tomas Mikolov et al., 2013)

Word2Vec was developed by Tomas Mikolov and team at Google.

Mikolov presented two architectures:

1) CBOW (Continuous Bag of Words): Predict word from context

- Input: [the, ___, sat, on] → Output: cat

2) Skip-gram: Predict context from word

- Input: cat → Output: [the, sat, on, mat]

- Trained on massive text corpora using simple neural networks

Mikolov Approach – Convert Text into Input-Output Pairs

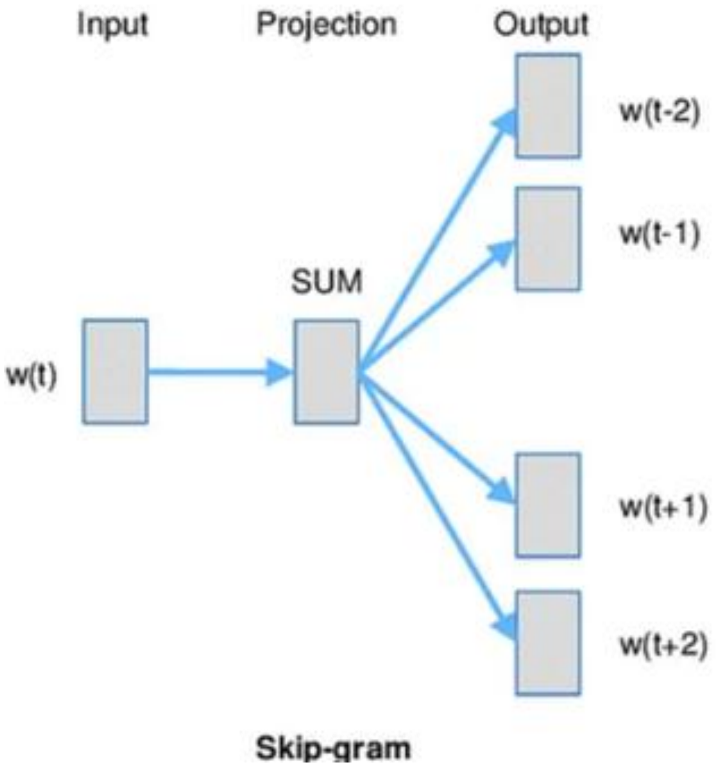
Suppose we have these sentences (corpus), we can convert them into input output pairs to be used for training a neural network:

Large language models are transforming business applications by introducing new levels of automation, intelligence, and personalization across industries. These models, trained on vast amounts of text data, can understand and generate human-like language, allowing businesses to enhance customer interactions, streamline operations, and gain deeper insights from data. In customer service, for instance, large language models power advanced chatbots and virtual assistants that can handle complex queries, provide 24/7 support, and adapt to customer needs in real time. In marketing, they help craft personalized messages, analyze consumer sentiment, and optimize content strategies with remarkable precision. Organizations are also using these models to automate report generation, summarize large documents, and even assist in writing code or drafting legal documents, significantly reducing time and cost. Moreover, they enable better decision-making by turning unstructured data—such as emails, reviews, or social media posts—into actionable business intelligence. As companies integrate these tools, they not only boost productivity but also redefine how humans and machines collaborate in the workplace. However, this transformation also raises questions about data privacy, ethical AI use, and workforce adaptation, making responsible deployment as important as technological innovation itself.

Skip-gram Architecture

- **Source Sentence:** "Large language models are transforming business applications"
- **Window Size:** 2 (2 words on each side of target word)
- **Task:** Predict context words from target word
- Given Target Word Predict → Context Words

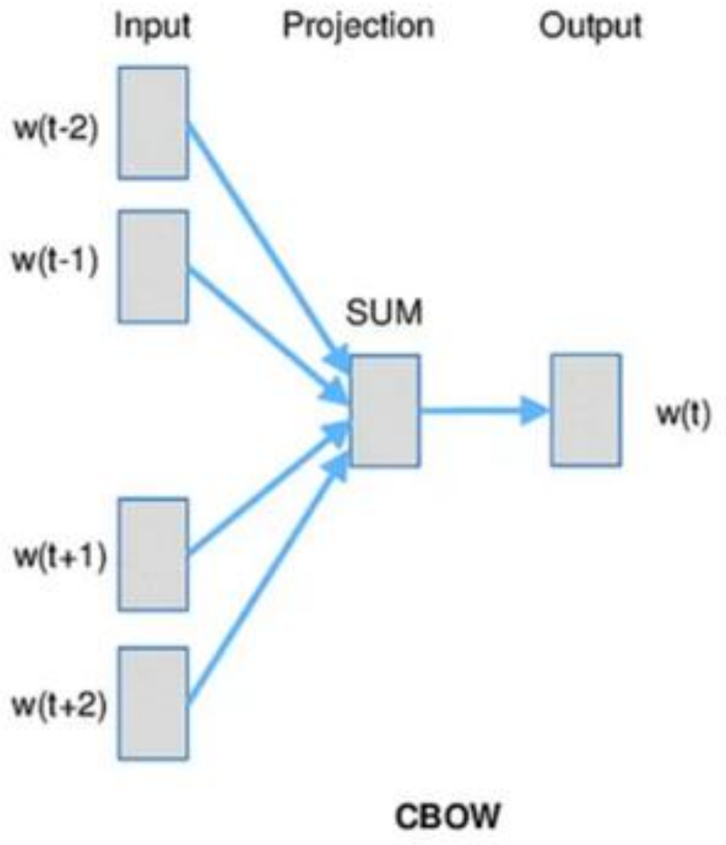
Input (Target Word)	Output (Context Words)
Large	[language, models]
language	[Large, models, are]
models	[Large, language, are, transforming]
are	[language, models, transforming, business]
transforming	[models, are, business, applications]
business	[are, transforming, applications]
applications	[transforming, business]



CBOW (Continuous Bag of Words) Architecture

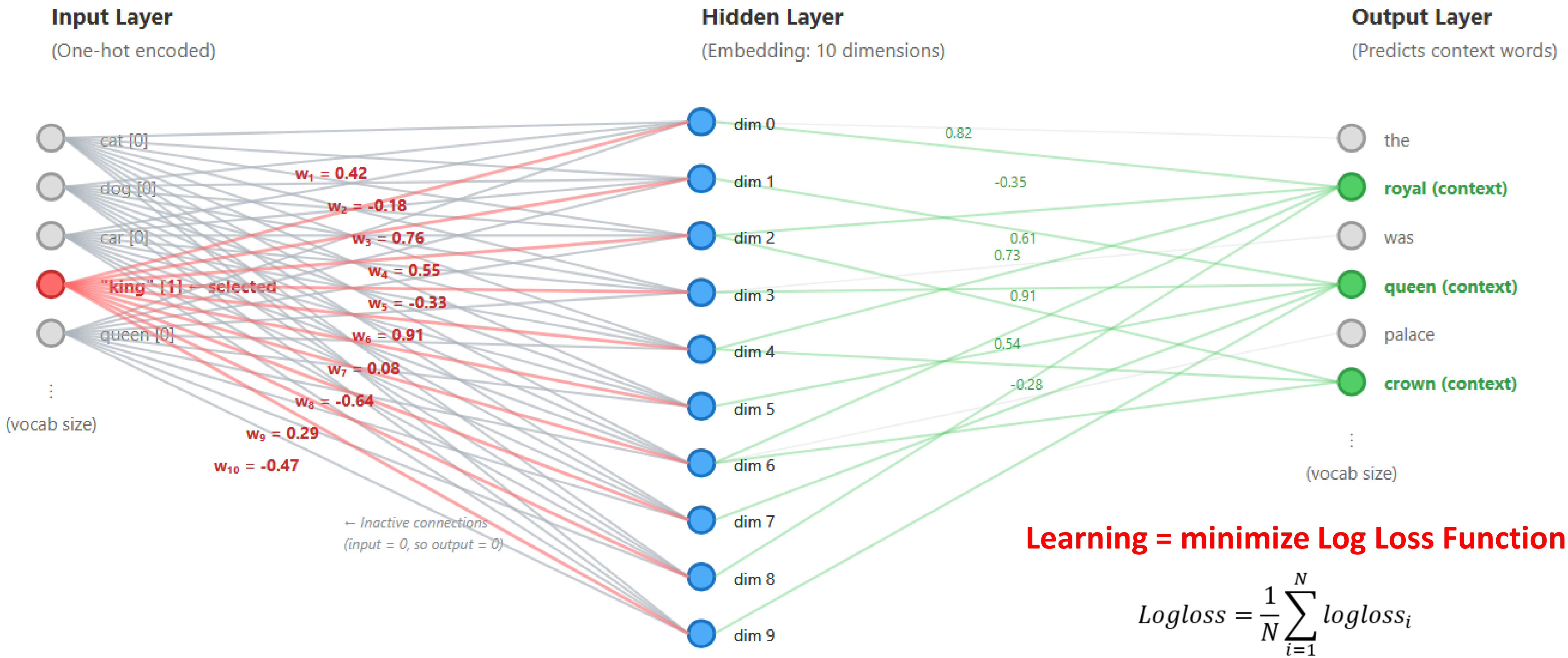
- **Source Sentence:** "Large language models are transforming business applications"
- **Task:** Predict target word from context words
- Given Context Words Predict → Target Word

Input (Context Words)	Output (Target Word)
[language, models]	Large
[Large, models, are]	language
[Large, language, are, transforming]	models
[language, models, transforming, business]	are
[models, are, business, applications]	transforming
[are, transforming, applications]	business
[transforming, business]	applications



Word2Vec Skip-gram: Learning Word Embeddings

Training sentence: "The **king** wore a royal crown, and the queen stood beside him"



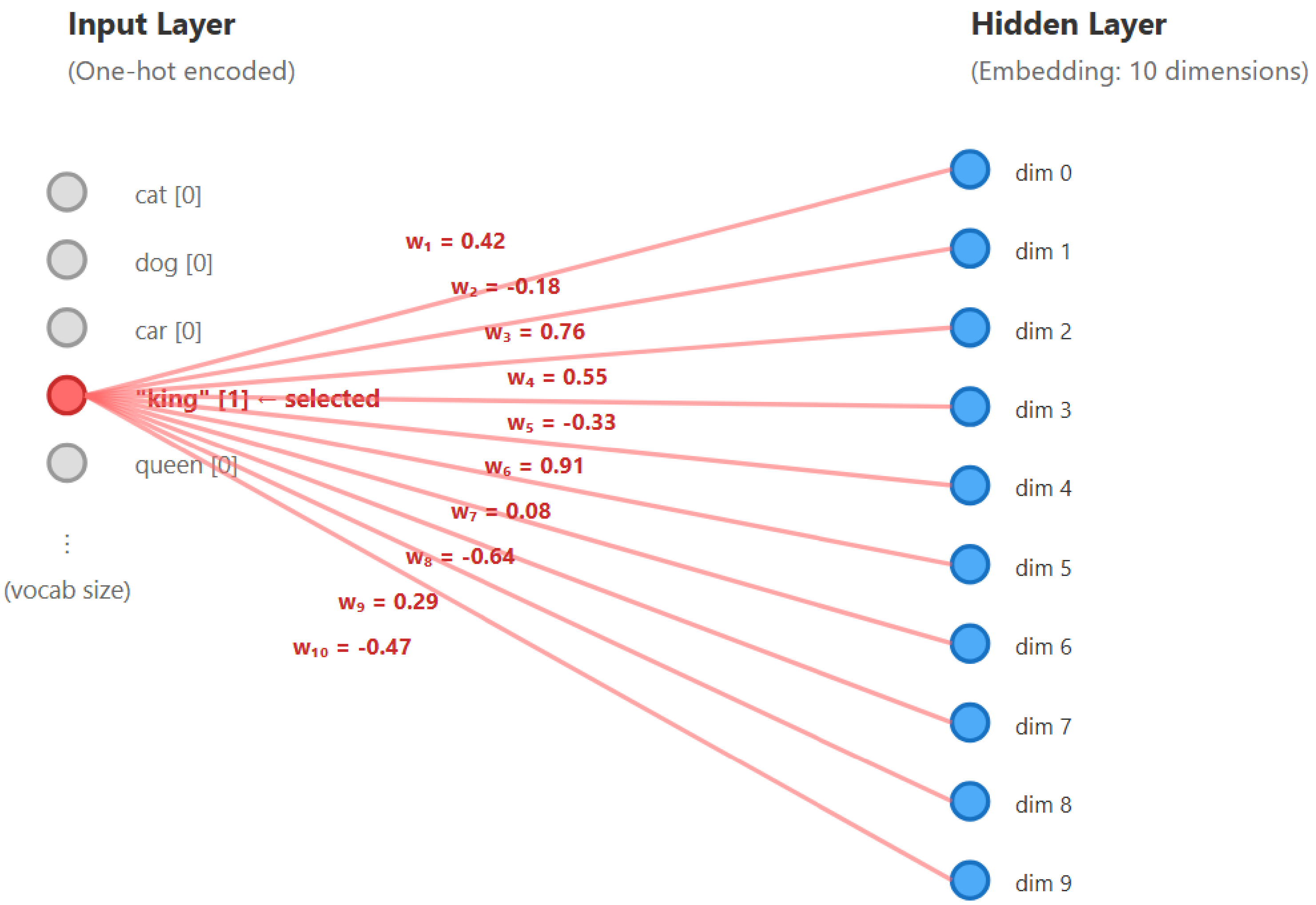
Learning = minimize Log Loss Function

$$Logloss = \frac{1}{N}\sum_{i=1}^{N} logloss_i$$

Training Process: Input Word "king" → Embedding → Predict Context Words

- Input→Hidden weights = Word Embeddings (e.g., king = [0.42, -0.18, 0.76, 0.55, -0.33, 0.91, 0.08, -0.64, 0.29, -0.47])
- Hidden→Output weights predict context: Given "king", maximize probability of "royal", "queen", "crown"
- Backpropagation updates both weight matrices → words in similar contexts get similar embeddings

Skip-Gram Model



Word2Vec Skip-gram: What's Really Happening?

The Story

We want to learn word vectors where similar words have similar vectors. The key insight: **words that appear in similar contexts are similar**. So we set up a prediction task.

For every word pair (focus word w, context word c) in our text, we ask: *how probable is it to see context word c near focus word w?*

$p(c|w) = \frac{\exp(v_c \cdot v_w)}{\sum_{c'} \exp(v_{c'} \cdot v_w)}$

The dot product $v_c \cdot v_w$ measures similarity. We use softmax to turn these scores into probabilities. Words that actually appear together should get high probability.

The Learning Process

We **maximize** the likelihood of all the (word, context) pairs we observe in our training text:

$\max_\theta \sum_{(w,c) \in D} \log p(c|w)$

This is softmax regression where we're classifying: "given focus word w, which context word c will appear?"

In practice, we flip this to **minimize the negative log likelihood** (standard loss function):

$\mathcal{L} = -\sum_{(w,c) \in D} \log p(c|w)$

As we optimize, word vectors adjust so that words appearing in similar contexts naturally end up close together in vector space.

Challenge: The denominator sums over the entire vocabulary (expensive!)
Solution: Negative sampling or hierarchical softmax

Word2Vec Skip-gram Model

Goal

Learn word vector representations by predicting **context words** from a **focus word**

The Probability Model

$$p(c|w;\theta) = \frac{\exp(v_c \cdot v_w)}{\sum_{c' \in C} \exp(v_{c'} \cdot v_w)}$$

- **Numerator**: Dot product $v_c \cdot v_w$ measures similarity between word vectors
- **Denominator**: Softmax normalization over all possible context words
- **Parameters** θ: All word vectors we're learning

The Optimization Problem

Maximize log likelihood over training data D:

$$\max_\theta \sum_{(w,c) \in D} \log p(c|w) = \sum_{(w,c) \in D} \left[v_c \cdot v_w - \log \sum_{c'} \exp(v_{c'} \cdot v_w) \right]$$

In practice: Minimize the **negative log likelihood loss**

$$\mathcal{L} = -\sum_{(w,c) \in D} \log p(c|w)$$

Key Insights

✓ This is **softmax regression** (multi-class classification where classes = all vocabulary words)

✓ Taking log turns products → sums, prevents underflow

✓ **Computational challenge**: Denominator requires summing over entire vocabulary for each prediction

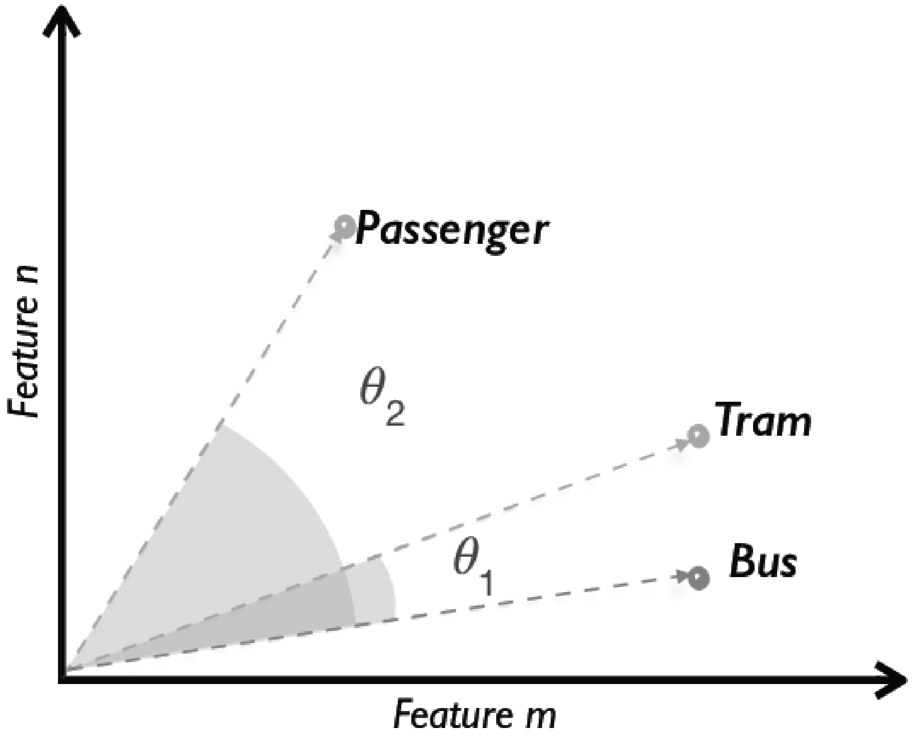
✓ **Solution**: Negative sampling or hierarchical softmax

NOT REQUIRED

Measuring Similarity Between Word Vectors

Why Compare Word Vectors?

- Word embeddings map words into a vector space.
- **Words with similar meanings** are placed **close together** in that space.
- To quantify this "closeness," we use **vector similarity**.



Cosine Similarity

Most common metric used to compare word vectors:

$$\text{cosine\_similarity}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$$

- Measures the **angle** between two vectors (not their magnitude).
- Ranges from **-1 to 1**:
  - 1 → Same direction (very similar)
  - 0 → Orthogonal (unrelated)
  - -1 → Opposite directions (very different)

Intuition

- Vectors for `"king"` and `"queen"` will have high cosine similarity.
- Vectors for `"apple"` and `"keyboard"` will have low similarity.

Use Pre-Trained Embeddings

Gensim is an open-source Python library used for **topic modeling** and **natural language processing (NLP)**.
It's great for working with **large text datasets** because it doesn't need to load all the data into memory at once.

Key features:

- Builds and uses **word embeddings** (like Word2Vec, FastText, Doc2Vec).
- Measures **semantic similarity** between words or documents.
- **Efficient and memory-friendly**, ideal for handling big collections of text.

```python
import gensim.downloader as api
from gensim.models import Word2Vec

# Load pre-trained Word2Vec model
word2vec_model = api.load("word2vec-google-news-300")


# Get vector for a word
cat_vector = model.wv['cat']
print("Vector for 'cat':", cat_vector[:5])  # Show first 5 dimensions

# Find similar words
similar_words = word2vec_model.most_similar('computer', topn=5)
print("Words similar to 'computer':", similar_words)

# Word analogies
result = word2vec_model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
print("king - man + woman =", result)
```

Open in Colab

Applications of Word Embeddings in NLP

1. **Semantic Similarity**
   Measure how similar two words, phrases, or documents are by comparing their vector representations.
   Example: Identifying that "doctor" and "physician" are closely related.
2. **Text Classification**
   Used as input features for tasks like spam detection, sentiment analysis, and topic classification.
   Embeddings provide rich, dense input for machine learning models.
3. **Named Entity Recognition (NER)**
   Help identify proper nouns and classify them into categories like person, location, or organization.
   Embedding-based models improve contextual understanding of named entities.
4. **Machine Translation**
   Map words from one language to another by aligning embeddings in multilingual space.
   Improves translation accuracy by leveraging semantic proximity.
5. **Question Answering & Chatbots**
   Used to understand queries and match them with appropriate answers or responses.
   Enable bots to interpret intent and context more accurately.

- **6. Information Retrieval**
  Enhance search engines by retrieving results based on semantic meaning, not just keyword matches.
  Example: Searching for “heart attack” returns documents containing “cardiac arrest.”